CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Matemáticas e Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

TRABAJO FIN DE GRADO

Árbol generador euclídeo de longitud y diámetros cortos

Autor: Inés Ruiz Rodríguez

Director: Manuel Abellanas

MADRID, JUNIO 2016

Dedicado a todas
aquellas personas que cazan dragones.
Muchas gracias Manuel por el apoyo y los consejos
y a mi familia por estar siempre ahí.

Resumen

En este trabajo se plantea el problema de conectar n puntos del plano mediante un grafo conexo que tenga tanto su longitud como su diámetro lo más pequeños posible.

Se consideran grafos geométricos cuyos vértices son puntos del plano y cuyas aristas son segmentos rectilíneos que tienen por extremos dos vértices. La longitud de un grafo es la suma de las longitudes de sus aristas. La distancia entre dos vértices del grafo es la suma de las longitudes de las aristas del camino más corto que los conecta. El diámetro de un grafo es la mayor de las distancias entre dos de sus vértices.

Está claro que el grafo de longitud mínima que conecta n puntos no contiene ciclos y, por tanto, se trata de un árbol. Si solo se considera esta condición, la solución la da el árbol generador euclídeo mínimo (EMST, Euclidean Minimum Spanning Tree en inglés). Sin embargo, el diámetro del EMST, como se verá en la memoria, puede ser muy grande. Por otra parte, si solo se considera la condición del diámetro mínimo, la solución sería el árbol generador de diámetro mínimo (MDST, Minimum Diameter Spanning Tree en inglés) del que se conocen, como en el caso del EMST, algoritmos eficientes para su cálculo. Este cómputo se desarrollará en SAGE, un entorno de cálculos matemáticos (MCE – Mathematics computing enviroment) de código abierto, y usará una estructura para definir grafos planos llamada DCEL (lista de aristas doblemente enlazada).

Al combinar las dos condiciones se plantea un problema de minimización biparamétrico para el que no se conocen soluciones. De hecho, no puede decirse que exista una solución óptima pues cada uno de estos criterios puede perjudicar al otro.

En el trabajo se empieza haciendo un estudio estadístico del diámetro del EMST de conjuntos aleatorios de puntos que confirma que, en general, el EMST no es el árbol de diámetro mínimo que conecta los puntos. A continuación, se emplea la triangulación de Delaunay y algunos de sus subgrafos notables - el grafo de Gabriel y el de vecindad relativa (RNG, Relative Neighbour Graph en inglés)- como técnica para buscar una solución de equilibrio entre longitud y diámetro cortos. La técnica consiste en calcular, para los mencionados grafos, el árbol generador de diámetro mínimo. Se completa el trabajo con el análisis experimental de las soluciones propuestas.

ABSTRACT

This work presents the problem of connecting n points in the plane by a connected graph that has both its length and diameter as small as possible.

Geometric graphs whose vertices are points in the plane and whose edges are rectilinear segments connecting vertices are considered. The length of a graph is the sum of the lengths of all edges. The distance between two vertices is the length of their shortest path in the graph. The diameter is the longest distance between any two vertices in the graph.

It is clear that the graph of minimum length that connects n points does not contain cycles and, therefore, is a tree. If only this condition is considered, the solution to the problem is given by Euclidean Minimum Spanning Tree (EMST). However, the diameter of EMST, as will be seen in the work, can be very large. Moreover, if only the condition of minimum diameter is considered, the solution would be the Minimum Diameter Spanning Tree (MDST) for which efficient algorithms that compute it are known, similar to the EMST case. This operation will be carried out in SAGE, a free open source Mathematics computing enviroment (MCE), and it will use a DCEL (Doubly Connected Edge List) structure to define planar graphs.

Combining the two conditions mentioned, a biparametric minimization problem with no known solutions arises. In fact, it cannot be stated that there exists an optimal solution, since the individual growth of one parameter may harm the other.

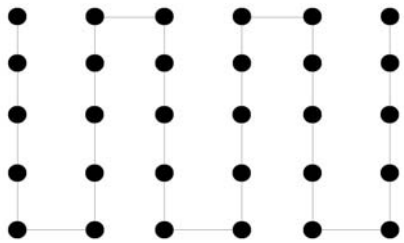
This work begins performing a statistical study about the diameter of EMST with random point sets that confirms that, generally, EMST is not the minimum diameter tree. Then, it uses the Delaunay Triangulation and some of its important subgraphs - the Gabriel Graph and the Relative Neighbour Graph (RNG) - as a technique to find a balanced solution between short length and short diameter. The procedure involves calculating, for the above graphs, the Minimum Diameter Spanning Tree. To conclude, it presents experimental tests of the proposed solutions.

Índice general

Capítulo 1

INTRODUCCIÓN

Este trabajo utiliza elementos de la Matemática Discreta y de la Geometría Computacional. Consiste en estudiar experimentalmente la longitud y el diámetro de un árbol geométrico, aquel grafo conexo y acíclico con vértices y aristas asociados a objetos geométricos, que conecte n puntos del plano encontrando aquel que tenga la longitud y el diámetro más cortos posibles. Es decir, es un problema multiparamétrico donde se debe minimizar la longitud, la suma de las longitudes de sus aristas, y el diámetro, distancia máxima entre dos vértices, de un árbol generador o de expansión, grafo con el mínimo conjunto de aristas, de puntos aleatorios.
Al tener dos variables con las que experimentar habría dos casos extremos: fijándose solamente en la longitud, el árbol de mínima distancia (EMST), o sólo en el diámetro, el árbol de mínimo diámetro. Como ya existen algoritmos eficientes para el EMST se comenzó por él. Pero no tenemos asegurado que este mismo árbol tenga el diámetro más pequeño y, por eso, se utilizó como base para empezar el estudio. Un ejemplo de EMST con un gran diámetro es:



Este trabajo tiene parte hecha en ordenador para luego poder hacer estudios estadísticos con la medida del diámetro y de la longitud en los distintos árboles que vayamos construyendo y sacar conclusiones a partir de estos resultados. La herramienta usada ha sido SAGE [1] porque es un entorno de cálculos matemáticos de código abierto basado en el lenguaje de programación Python. También incorpora librerías de visualización de gráficos fáciles de usar.

El objetivo es, con la ayuda del ordenador, estudiar la longitud y el diámetro de árboles geométricos de extensión y alcanzar unos valores satisfactorios, en la

longitud y el diámetro, dependiendo de a que parámetro le damos más importancia en distintas hipotéticas situaciones. Esto viene a decir que existen grafos con un diámetro mejor que el de un posible EMST pero el valor de la longitud empeoraría.

Capítulo 2

EMST

En este capítulo se desarrollará la creación del Árbol Euclídeo de Mínima Distancia, los elementos utilizados y los primero cálculos del diámetro y de la longitud.

2.1. Definiciones básicas

Antes de empezar a implementar los algoritmos para el EMST se deben explicar algunos conceptos básicos de la Matemática Discreta.

Definición. Un grafo [2] o grafo simple es un par (V,A), donde el conjunto finito V tiene miembros llamados vértices o nodos y la familia finita A de pares no ordenados de vértices tiene elementos llamados aristas o arcos. Se denota como $G = (V, A)$.

Definición. El grado de un vértice de un grafo G es el número de aristas o arcos que parten de él.

Definición. Los grafos no dirigidos [3] tienen aristas no orientadas, $(v, w) = (w, v)$, y los grafos dirigidos están compuestos de aristas con dirección ($(v, w) \neq (w, v)$).

Definición. Un grafo etiquetado [3] tiene por cada arista y/o vértice un valor asociado.

Definición. Un grafo ponderado [3] (grafo con pesos) es un grafo etiquetado en el que existe un valor numérico asociado a cada arista o arco.

Definición. Un grafo conexo [4] $G = (V, A)$ tiene $\forall$ par u,v ϵ V un camino en el grafo que los conecta.

Definición. T es un árbol si es un grafo conexo y acíclico, es decir, no tiene ciclos o caminos en el grafo que salgan y vuelvan al mismo vértice sin repetir aristas.

Definición. Las hojas de un árbol T son aquellos vértices que tienen grado 1.

Definición. Un árbol geométrico [5] (grafo geométrico) es aquel cuyos vértices y aristas están asociados a objetos geométricos. Entre todas las posibles representaciones nos quedaremos con los elementos de un espacio euclídeo (grafo euclídeo) donde los vértices son puntos del plano y las aristas son segmentos rectilíneos que unen nodos. El peso de una arista es su longitud.

Definición. La distancia euclídea se calcula mediante la fórmula:

$$dist(u, v) = \sqrt{(u2 - u1)^2 + (v2 - v1)^2}$$

Definición. La distancia en un grafo, $d(u, v)$, es la longitud del camino más corto entre u y v; siendo un camino una secuencia de una o más aristas que conectan dos vértices.

Para concluir con éste apartado de descripciones se terminará con la explicación del elemento básico del proyecto pero que requiere de las anteriores de definiciones. Con él podremos obtener una longitud pequeña ya que este parámetro depende de la cantidad de aristas usadas en el grafo G.

Definición. El árbol generador (de expansión o recubridor) [6] T, de un grafo G conexo y no dirigido, está compuesto por todos los vértices y algunas aristas de G de forma que T es conexo y acíclico.

2.2. DCEL

En la parte computacional se usará el DCEL [5] (lista de aristas doblemente enlazada). Es una estructura de datos muy utilizada en Geometría Computacional para manipular eficientemente los elementos topológicos asociados con los objetos de un grafo (vértices, aristas, caras). Se usa exclusivamente en grafos planos o subdivisiones del plano.

Definición. Un grafo plano o planar es aquel grafo representado en el plano sin ninguna arista cruzando o cortando otra.

La estructura se compone de 3 sublistas:

1. Vértices. Se descompone en 2 sublistas que indican el punto en el plano, con su componente x y su componente y, y el índice en la lista de aristas para indicar un arco que empieza desde ese vértice.

2. Aristas. Guarda 2 semiaristas orientadas de un arco del grafo con misma dirección y sentidos contrarios. Cada una de estas semiaristas se divide en:

- Índice del vértice origen.
- Índice de la semiarista gemela o de sentido contrario.
- Índice de la anterior semiarista.
- Índice de la siguiente semiarista.
- Índice de la cara que deja a su izquierda si recorrieramos la arista en la dirección que indica esta semiarista.

3. Caras. Contiene un índice de una arista de su borde.

Será la base para la creación de grafos y árboles del trabajo.

2.3. Triangulación de Delaunay

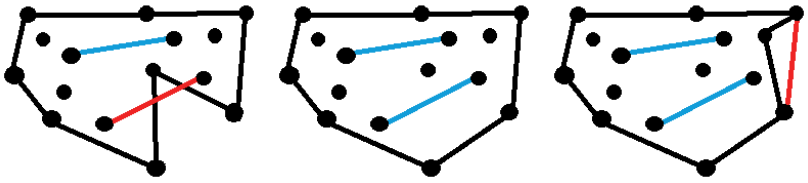
Sabiendo la herramienta computacional a utilizar se debe minimizar el tiempo de computo o la complejidad de los algoritmos por venir. Se sabe que algunos de ellos utilizan otros grafos como base del problema; se deberá tomar uno que sea rápido de calcular, crear y utilizar. El más fácil de calcular es el grafo completo $G = (V, A)$: $\forall\ v\ \epsilon\ V,\ grado(v) = |V| - 1$. Crearlo no sería complicado pero utilizarlo en los algoritmos llevaría mucho tiempo por la gran cantidad de aristas que contendría y cuantos más puntos se añadan al problema, muchas más aristas se sumarían al grafo. No se podría usar la estructura DCEL porque el grafo completo no es un grafo plano.

Pensando en el futuro algoritmo del EMST se puede observar una propiedad de este árbol:

$$\boldsymbol{T.\ Delaunay \supset EMST}$$

Definición. Un conjunto de puntos del plano es convexo si el segmento que une dos puntos está contenido en él.

Definición. El cierre convexo de un conjunto es el menor conjunto convexo que lo contiene.



(a) No c. convexo (b) C. convexo (c) No c. convexo

La triangulación de Delaunay es la triangulación más conocida porque tiene una gran cantidad de usos y resuelve multitud de problemas.

Definición. La triangulación de una nube de puntos C es un conjunto de triángulos cuyos vértices pertenecen a C tal que su unión es igual al cierre convexo de C y la intersección de dos de ellos es uno de sus lados o bien uno de sus vértices o bien es vacía.

"*Sea S contenido en el plano como un conjunto finito de puntos que no forman una línea recta. Decimos que T es una Triangulación de Deloné si T es una triangulación del cierre convexo del conjunto S, con S como conjunto de vértices de esa triangulación y que además, las circunferencias circunscritas a los triángulos de T no contienen a ningún punto de S en su interior.*" [7]

Algunas características de la triangulación de Delaunay [8] son:

- Es un grafo conexo.
- Los triángulos se forman con las aristas que unen a sus vecinos. Son vecinos si existe un círculo que contiene a esos dos puntos y a ningún punto del conjunto.
- Tres puntos de la nube de puntos son vértices de un mismo triángulo si y sólo si la circunferencia circunscrita que forman esos puntos no contiene otros puntos de la nube en su interior.
- Dos puntos de la nube describen una arista de la triangulación si y sólo si la circunferencia que los define no contiene otros puntos de la nube en su interior.
- Las aristas de la triangulación de Delaunay son aristas duales de las aristas del diagrama de Voronoi.

Definición. *"[...] un diagrama de Voronoi es una subdivisión del plano (en el que se encuentran n puntos) en n regiones, de forma que cada segmento de línea que sirve de división entre dos regiones es equidistante a los dos puntos de sendas regiones (y, por tanto, perpendicular al eje que une esos dos puntos)."* [8]

El diagrama de Voronoi, al igual que la triangulación de Delaunay, tiene incontables aplicaciones en la vida real y sobre campos muy diferentes.

2.4. Árbol generador mínimo euclídeo

El árbol generado mínimo euclídeo, EMST(P) [9], de un conjunto P de n puntos es un grafo conexo cuyos vértices son los puntos del conjunto P que minimiza la suma de las longitudes de las aristas. Tiene como pesos en las aristas la distancia euclídea de sus extremos.

Definición. La longitud en un árbol es la suma de los pesos de todos sus arcos.

En general este tipo de estructuras son normalmente usadas para ahorrar recursos, siendo su aplicación más común la instalación de redes.

2.4.1. Algoritmo de Prim

Permite calcular un árbol generador mínimo en un grafo ponderado conexo y no dirigido.

El algoritmo de Prim [5, 9] comienza en un vértice e incrementa continuamente el tamaño de un árbol agregándole vértices mediante aristas, las cuales deben tener el mínimo peso entre todas ellas. Es decir, en cada paso, vamos añadiendo la arista de mínimo peso que conecta a un vértice todavía no perteneciente al árbol con otro que sí está. Se finaliza cuando no quedan más vértices por agregar.

En la parte computacional se utiliza el DCEL para crear otra estructura a partir de él. Es una lista donde cada elemento de ella es otra lista que contiene en su primera componente el índice de un vértice según la estructura DCEL anterior; y en su segunda componente están los índices de los vértices según esta estructura con la distancia euclídea entre los dos vértices.
El pseudocódigo del algoritmo de Prim en esta estructura es:

```
def Prim( DCEL ):
    T = estructuraDistancias( DCEL )
    prim = [ [ primer vertice, [ ] ] ]
    # prim también tendrá la misma estructura que T
    V = [ primer vertice ] # para controlar si tenemos todos los vertices
    nodoEncontrado(primer vertice, ListaAr, V, prim, T)
    # Esta función se comunica con la nueva estructura
    # Encuentra todas las posibles aristas para añadir
    # Esas aristas se añaden a ListaAr
    # El primer vértice se añade a prim
    while numeroVertices != len(V):
        ind = 0
        min = INFINITO
        for i in ListaAr:
            if min < distancia en i:
                min = distancia en i
                ind = indice de i en ListaAr
        nodoEncontrado(ind, ListaAr, V, prim, T)
    return prim
```

Existe otro algoritmo para calcular el EMST: el algortimo de Kruskal. Consiste en, dado un grafo conexo, no dirigido y ponderado, seleccionar las aristas (todavía no pertenecientes al árbol) de menor peso siempre que no cree ningún ciclo. Se finaliza cuando se hayan seleccionado $|V| - 1$ aristas. El algoritmo de Kruskal tiene como orden de complejidad computacional temporal $O(alogn)$, siendo a el número de arista del grafo y n el número de vértices. Éste orden de complejidad se debe a la ordenación de las aristas de menor a mayor peso. Mientras que la complejidad del algoritmo de Prim es del orden $O(n^2)$.
En un grafo denso, el algoritmo de Kruskal a tiende a $n(n-1)/2$ aristas y su orden de complejidad sería $O(n^2logn)$. Por lo tanto, el algoritmo de Prim tiene mejor complejidad en grafos densos. Es decir, para un grafo de pocas aristas es mejor usar Kruskal y para un grafo de muchas aristas es mejor el algoritmo de Prim.

Para comprobar la eficiencia del árbol EMST creado a partir de una estructura particular se han utilizado otras estructuras implementadas en unas librerías de SAGE:

```
from scipy.sparse import csr_matrix
from scipy.sparse.csgraph import minimum_spanning_tree
import scipy.spatial
```

Creando dos EMST con los dos distintos métodos, con la misma nube de puntos, se comprobó su veracidad y eficiencia.

Podría decirse que ya se tendría uno de los objetivos completados: obtener un árbol generador de longitud mínima. Pero, ¿el diámetro también es el mínimo? No podemos asegurar que sea el mínimo y eso nos lleva a estudiar otras estructuras sabiendo que si queremos mejorar el diámetro (longitud) aumentará la longitud (diámetro) del grafo.

2.4.2. Diámetro

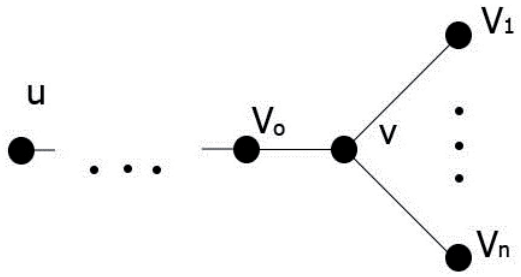
El segundo parámetro a estudiar es el diámetro.

Definición. El diámetro en el grafo es la mayor distancia entre dos pares de vértices.

Los caminos trazados en el árbol o grafo pueden medirse por cantidad de aristas recorridas o por las distancias euclídeas. En este trabajo, el diámetro será la máxima distancia entre dos vértices del grafo, siendo ésta la suma de las longitudes de las aristas del camino mínimo que los conecta. La primera forma de hallar el diámetro, que se implementó en SAGE, fue obtener todas las hojas del EMST y, con posterioridad, calcular la distancia entre todos los pares de vértices de esa lista. El diámetro

sería la mayor de todas ellas.

Demostración: Suponemos el diámetro $D = dist(u, v)$, $u \in H$, $v \notin H$, siendo H la lista de hojas del árbol. Una hoja de un árbol es aquella que está conectada a un solo vértice. Por lo tanto v está unido a más de un vértice que forma una lista V de vértices enlazados con él. Existe un vértice $V_0 \in V$ perteneciente al camino que une v con u, entonces, hay $V - \{V_0\}$ vértices con los que u puede conectar y obtener $dist(u, V_i) = dist(u, v) + dist(v, V_i)$, $i = 1, 2, ..., n$. Contradicción, v debe ser una hoja.



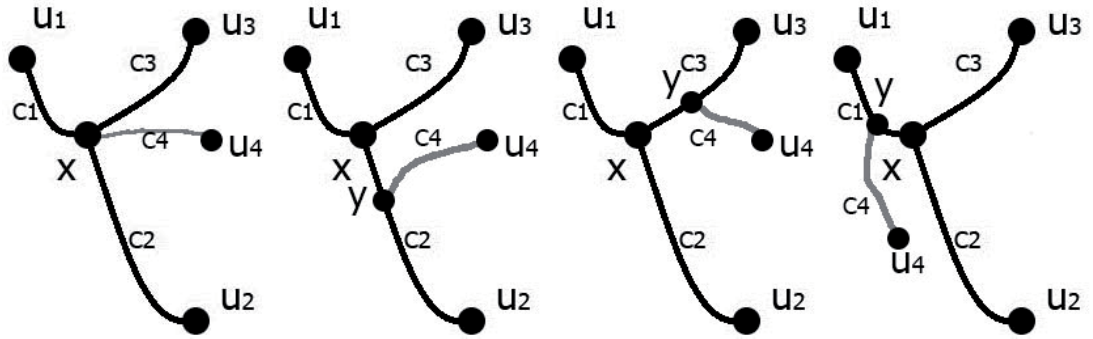
Es un método más eficiente que el algoritmo de fuerza bruta. Pero, aún así, sigue siendo muy costoso computacionalmente.

Siguiendo con la idea de usar una lista de hojas del árbol se encontró otro algoritmo para calcular el diámetro [10]. Consiste en tomar un vértice $u1 \in H$, siendo H la lista de hojas del árbol. Hacer todos los caminos de $u1$ al resto de elementos de H y quedarnos con el vértice $u2$ más alejado. Repetimos el paso anterior pero con $u2$ y, entonces, obtenemos otro vértice $u3$ siendo $dist(u2, u3)$ las mayor de todas y el diámetro del árbol. Código:

```
def diameterTree(Tree):
    T = deepcopy(Tree)
    a = 0
    # d = valor del diametro, c = camino
    d,c=caminoSum(a,T)
    a = c[len(c)-1]
    d,c=caminoSum(a,T)
    return d
```

Demostración: Se tiene que comprobar que $dist(u2, u2)$ es el diámetro del árbol. En primer lugar se sabe que, por el algoritmo, $dist(u1, u2) < dist(u2, u3)$ y ambos caminos tienen más de un vértice en común, porque si no tuvieran alguno habría una contradicción ya que los dos tienen al vértice $u2$; y deben tener más de uno porque $u2$ es una hoja, sólo se comunica con un vértice. Entonces ese único vértice x pertenecerá a los dos caminos. Llamaremos $C1 = dist(x, u1)$, $C2 = dist(x, u2)$ y

$C3 = dist(x, u3)$. No puede existir ningún camino maximal que no contenga o a $u1$ o $u2$ o $u3$ ya que el algoritmo recorre en profundidad todo el árbol y si existieran caminos más grandes el algoritmo los habría encontrado.



(a) Caso 1 (b) Caso 2 (c) Caso 3 (d) Caso 4

Suponemos que el diámetro es $dist(u2, u4)$. Como se ve en la figura del *Caso 1* observamos que $C3 < C4$. Contradicción. El algoritmo en la segunda búsqueda en profundidad nos habría dado ese camino antes que el camino de x a $u3$. Ocurre lo mismo con los caminos desde $u1$. La figura del *Caso 3* también sería contradictoria por la explicación anterior. $C3 - dist(x, y) < C4$, si fuera así el algoritmo habría devuelvo $u4$ y no $u3$. Aunque los *Casos 2* y *4* parezcan distintos al final es lo mismo si juntamos x e y en un mismo punto y le sumamos $dist(x, y)$ a C4 y también en el $Caso2$ a C2 y en el $Caso4$ a C1.
Llegamos a las siguientes conclusiones:

- Todos los casos pueden expresarse como en la figura $Caso1$.
- Diámetro $\equiv dist(u2, u4)$ no es posible porque el algoritmo lo habría encontrado.
- Diámetro $\equiv dist(u1, u4)$ sería incorrecto porque al hacer la búsqueda en profundidad desde $u1$ se obtendría $u4$ antes que $u2$ por las distancias.
- Diámetro $\equiv dist(u3, u4) = C3 + C4$ sería el menos probable por lo siguiente: $C4 < C3 < C2$. Porque:

$$\left.\begin{array}{l} C1 + C2 < C2 + C3 \longrightarrow C1 < C3 \\ C1 + C3 < C1 + C2 \longrightarrow C3 < C2 \end{array}\right\} \Rightarrow C1 < C3 < C2$$

$$\left.\begin{array}{l} C1 + C4 < C1 + C2 \longrightarrow C4 < C2 \\ C2 + C4 < C2 + C3 \longrightarrow C4 < C3 \end{array}\right\} \Rightarrow C4 < C3 < C2$$

La primera inecuación expresa el resultado de la segunda búsqueda en profundidad desde $u2$ ya que el punto más alejado de $u1$, $u2$, encontró un vértice con un camino

más largo, $dist(u2, u3) = C2 + C3$.
La segunda inecuación sale de que en la primera búsqueda se obtuvo de $u1$ a $u2$ antes que $u3$ entonces $dist(u1, u3) < dist(u1, u2)$. Las dos últimas inecuaciones son resultado de todo lo anterior.

Calcular el diámetro tiene una complejidad computacional temporal lineal, es decir, del orden $O(n)$ siendo $n = |V|$.

El método usado para recorrer el árbol y buscar todos los posibles caminos en el árbol con sus respectivas distancias es un algoritmo recursivo basado en la búsqueda en profundidad.

Definición. La búsqueda en profundidad (Depth First Search) es un algoritmo que recorre un grafo de forma ordenada. Consiste en ir expandiendo todos los vértices localizados en un camino concreto. Cuando ya no se puede continuar, regresa (Backtracking) y repite el mismo proceso con cada uno de los vecinos de los vértices ya procesados.

Se quiere encontrar el camino más largo entre las hojas u y v. Se empieza en una hoja u y se añade a una lista $S1$, el resto de vértices a una lista $S2$:

1. Se obtienen los vecinos del nuevo vértice añadido a $S1$ que no estén en $S1$ y forman el nuevo conjunto Vec.

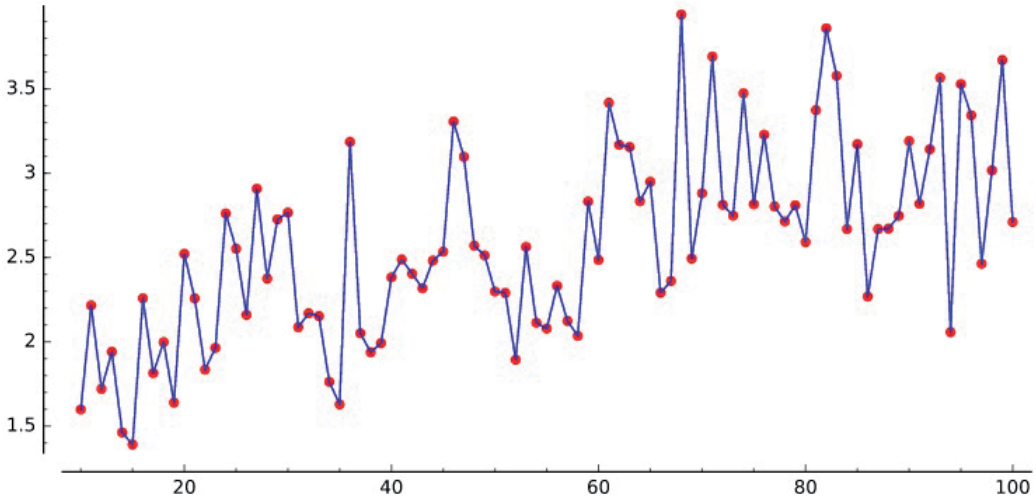
2. Si Vec está vacío (devuelve un valor que simbolice un final de camino, por ejemplo un -1) o contiene a v se termina la rama de ejecución (devuelve el valor de la distancia del camino).

3. En otro caso, a cada elemento de Vec (individualmente, creando distintas ramas de ejecución): se añade a $S1$, se borra de $S2$ y volvemos al paso 1.

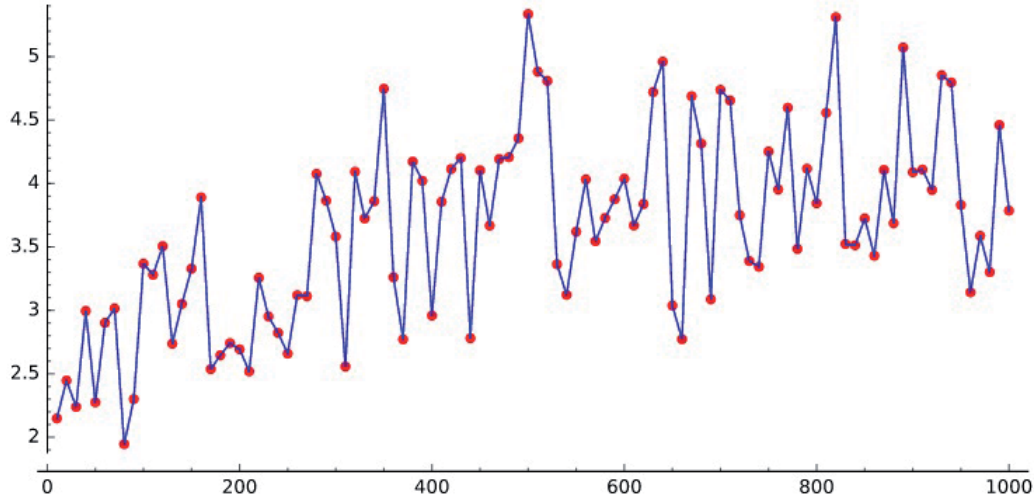
Cada vez que el algoritmo recursivo devuelva un valor habrá que tener en cuenta si es un final de camino sin encontrar a v o encontrándolo. Si se encuentra no habrá necesidad de continuar buscando más caminos.

2.4.3. Resultados del diámetro

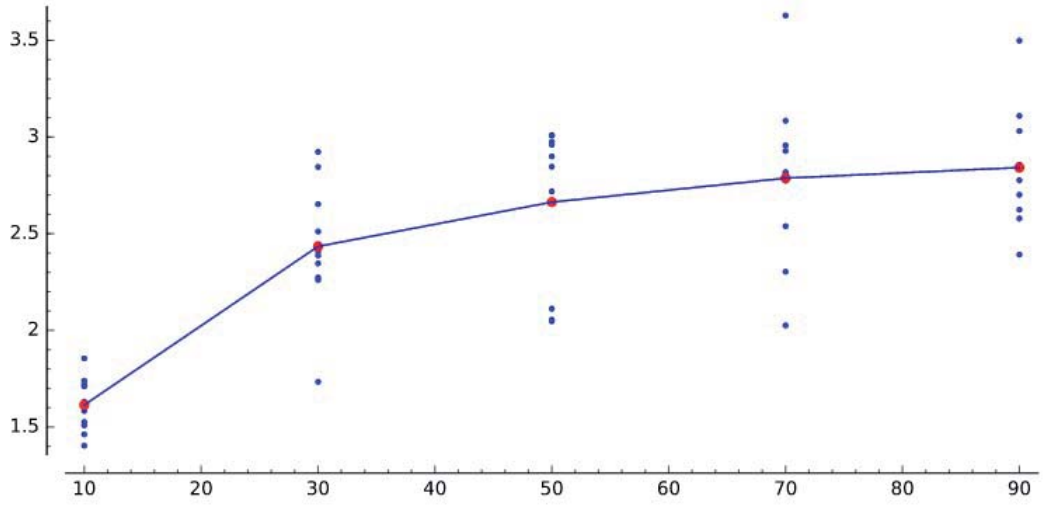
Con un algoritmo eficiente para obtener el EMST y otro para calcular el diámetro se procede a observar los resultados obtenidos de una gran cantidad de puntos aleatorios en el cuadrado unidad con una distribución uniforme.



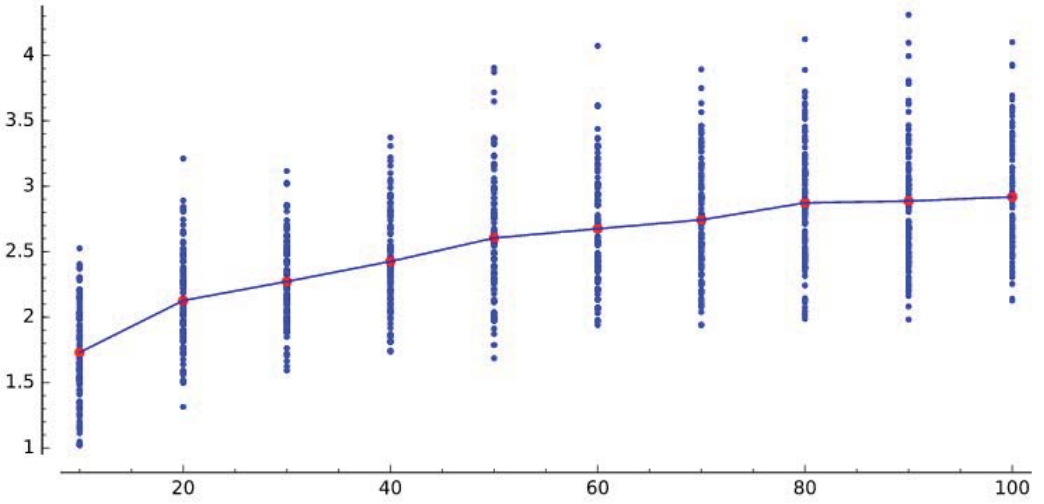
La gráfica anterior tiene por eje x o de abscisas la cantidad de puntos usados para los cálculos, y en el eje y o de ordenadas el valor del diámetro. Como se puede observar, no podemos sacar ninguna conclusión a partir de esta gráfica. Los valores obtenidos no siguen ningún patrón; la diferencia entre un valor con su anterior o con su consecutivo no predetermina otro valores. El único detalle a destacar es que, poco a poco, el diámetro tiene un valor más grande acorde aumenta la cantidad de puntos que empleamos.



Ahora se puede ver mejor un intervalo de crecimiento del diámetro. Pero esta información no es suficiente porque, ¿los valores con una determinada cantidad de puntos se comporta igual? ¿Se acumulan en un punto o se dispersan por toda la recta del eje y o forman un intervalo con un máximo y un mínimo?



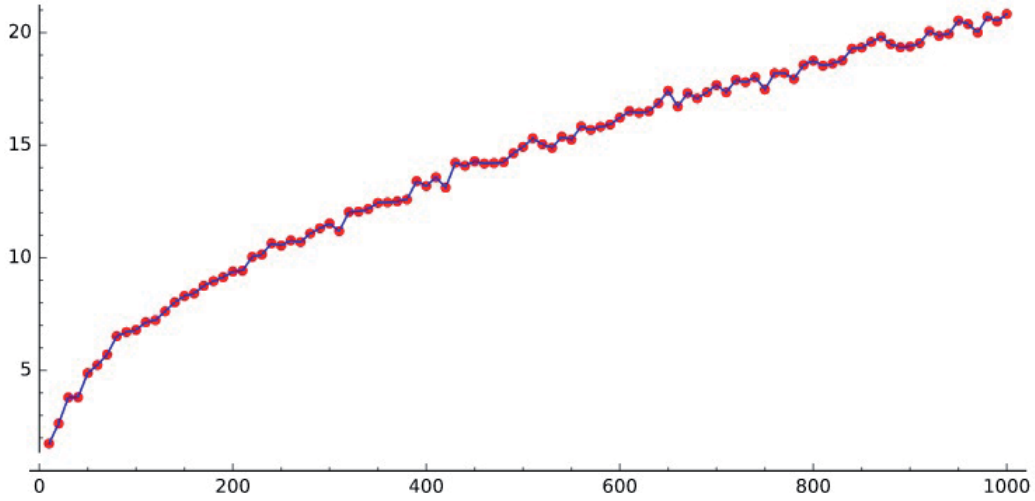
En esta gráfica se calculó n veces el diámetro con una cantidad m de puntos aleatorios distintos a todas esas n veces. Los puntos azules son los valores calculados y los puntos rojos son la media de los resultados en cada m cantidad de puntos del plano. Los puntos no se acumulan en la media pero se puede observar una tendencia a no alejarse mucho, creando un intervalo.



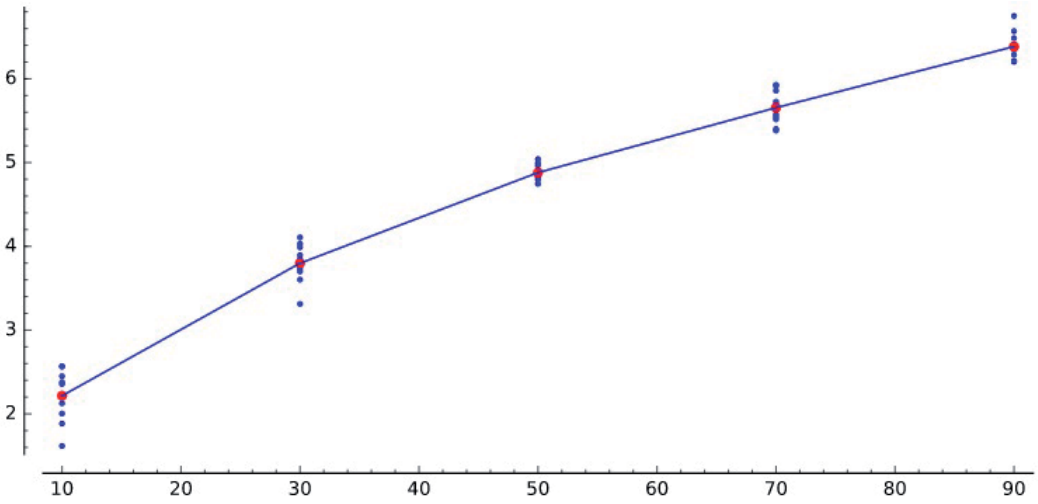
Ya se puede asegurar que existe un intervalo $[media - \epsilon, media + \epsilon]$ de valores en el que cae el diámetro con una alta probabilidad y, a medida que aumenta la cantidad de puntos, este intervalo aumenta un δ a su máximo y mínimo.

2.4.4. Resultados de la longitud

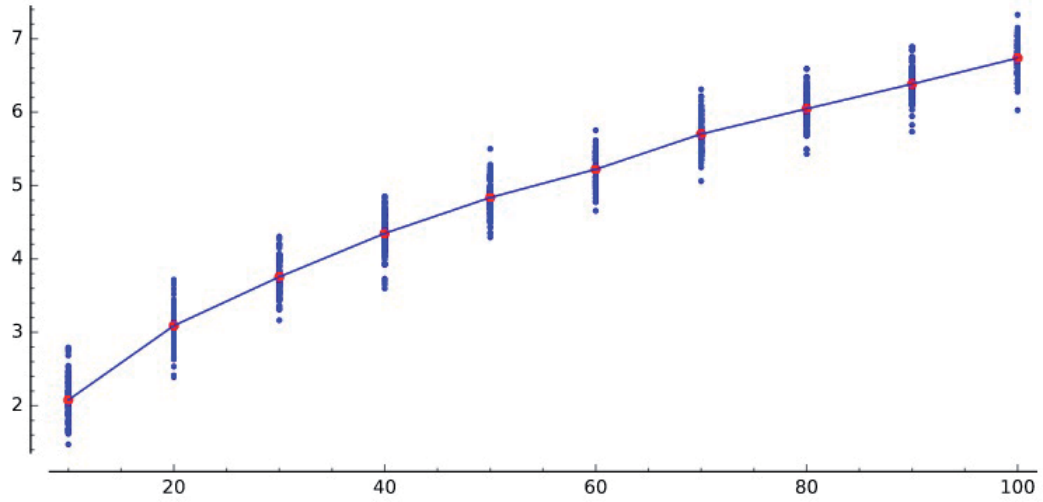
También se estudiará el comportamiento del parámetro longitud con la misma metodología usada en el diámetro.



La longitud es más sensible a la cantidad de puntos utilizados y no tiene un crecimiento tan caótico como el del diámetro pero alcanza valores más altos.



Con unas pocas iteraciones se observa una agrupación de puntos alrededor de la media de estos. Por lo tanto, la media y la mediana son cercanas.



Las conclusiones de esta gráfica son las mismas que en el diámetro: existe un intervalo $[media - \epsilon, media + \epsilon]$ de valores donde la longitud está con una alta probabilidad. La longutid tiene como $mediana = media \pm \alpha$, siendo α un número cercano o igual a 0.

Capítulo 3

MDST

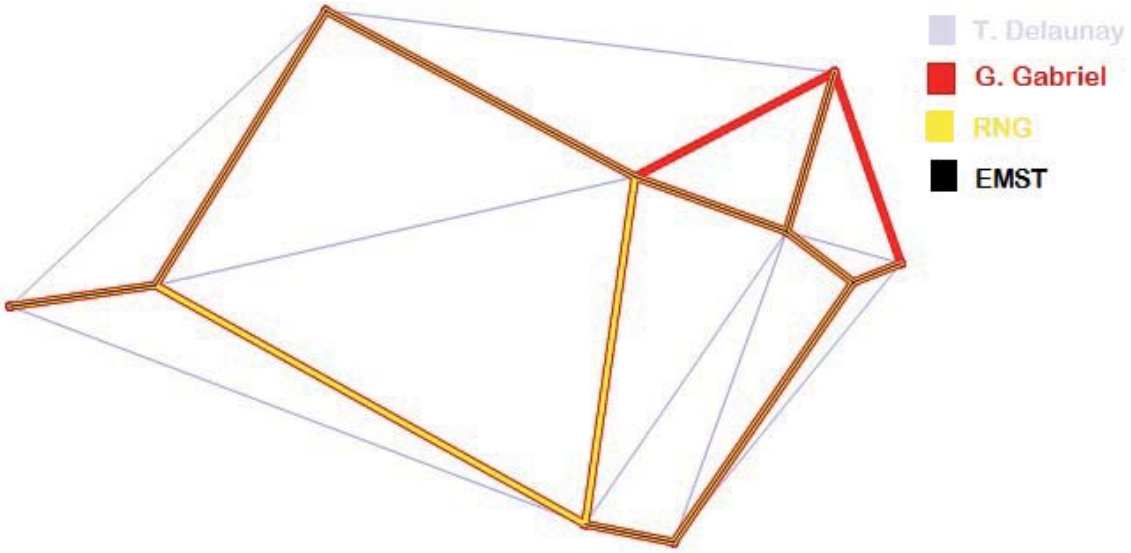
Al ser una investigación, se puede desarrollar con múltiples posibilidades e ideas, las cuáles, pueden converger en un resultado o conseguirnos más preguntas a resolver.

En un principio se iba a comenzar intercambiando aristas del árbol de mínima longitud lo que empeoraría la longitud pero, ¿mejoraría el diámetro? ¿Cuál de todas las aristas habría que cambiar? ¿Daría igual la arista a elegir o habría que hacer un estudio previo de selección?
Pero no es un sólo problema pues al intercambiar aristas habría que modificar dos de ellas, siempre conservando las propiedades del árbol o de la triangulación.

Se comenzó a dudar sobre la efectividad de esta idea hasta que se encontró la tesis de Wu y Chao [10] sobre árboles de expansión. Explican un algoritmo, desarrollado más adelante en la memoria, para calcular el MDST: árbol generador de mínimo diámetro. Este algoritmo requiere de un grafo del que partir. Por ello, se volverá a usar la triangulación de Delaunay y una propiedad muy importante y, de la cuál, obtendremos más grafos:

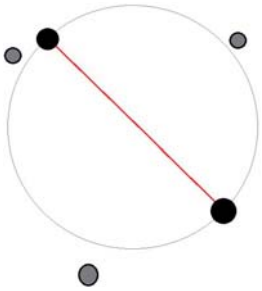
$$\textbf{\textit{T. Delaunay}} \supset \textbf{\textit{G. Gabriel}} \supset \textbf{\textit{RNG}} \supset \textbf{\textit{EMST}}$$

La idea de usar el MDST es de que, a partir de un grafo conocido, podemos obtener un árbol (para minimizar la longitud pues tendrá menos aristas) con el diámetro más pequeño posible; y observar la medida de la longitud y del diámetro de una nube de puntos aleatorios en el plano.



3.1. Grafo de Gabriel

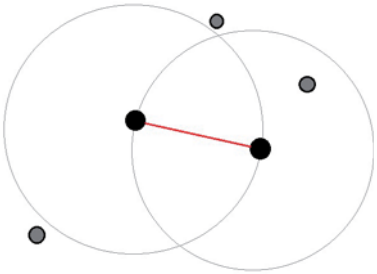
El grafo de Gabriel [5] es un subgrafo conexo de la triangulación de Delaunay y sus aristas deben cumplir que la circunferencia, con diámetro la longitud del arco y definida por los extremos de la arista, no contenga otros vértices del grafo.



En SAGE se usó la estructura DCEL como base del algoritmo. A partir de la triangulación de Delaunay se fué descartando aquellas aristas que no cumplieran con la anterior propiedad del grafo de Gabriel.

3.2. Grafo de vecindad relativa

El grafo de vecindad relativa (RNG) [5, 12] es un subgrafo conexo del grafo de Gabriel; entonces también es subgrafo de la triangulación de Delaunay. Las aristas de este grafo cumplen con la propiedad de vecinos relativos: la lente intersección, creada por las circunferencias con centro en los extremos del arco y radio la longitud de la arista, no contiene vértices del grafo.



El primer algoritmo computacional creado usó el grafo de Gabriel como elemento base y otro la triangulación de Delaunay. En los dos se partía de un grafo conexo al cuál se le borraban las aristas que no cumplían la propiedad de vecinos relativos. Es verdad que el grafo de Gabriel tiene menos arcos que la triangulación de Delaunay pero el cálculo previo al grafo de Gabriel hacía más lento este primer algoritmo. Por ello, se optó por usar la triangulación de Delaunay como base para la obtención del grafo de vecindad relativa.

3.3. Desarrollo del MDST

Como ya se dijo anteriormente, el código a continuación se encontró en el libro de Wu y Chao [10]:

```
def MDST( G, DCEL ):
    SPL = ShortestPathLength(G,DCEL)
    # en SPL se guardan TODAS las distancias de un vértice a otro
    lamda = Infinity
    for i in G:
        u1 = D[1][i][0] #ind de Vertice
        u2 = D[1][gemela(i,D)][0] #ind del otro Extremo de la arista
        while v < len(D[0]):
            d = SPL[v][u1][0] - SPL[v][u2][0]
            V.append([d,v]) # (diferencia, ind del vertice)
        #Ordenamos por los resultados de la diferencia anterior
        V.sort()
        while j < len(V)-1:
            if V[j][0] < V[j+1][0]:
                V1 = { v_j | j <= i }
                V2 = { v_j | j > i }
                dgU1 = DG(u1,V1,SPL,DCEL)
                dgU2 = DG(u2,V2,SPL,DCEL)
                d = dgU1+dgU2+SPL[u1][u2][0]
                if d < lamda:
                    lamda = d
                    x = u1
                    y = u2
```

```
            W1 = V1
            W2 = V2
            vert = i
    T1 = Dijkstra(G,x,W1,SPL,DCEL,y)
    T2 = Dijkstra(G,y,W2,SPL,DCEL,x)
    T = noRepetidos(T1,T2+[vert],SPL,DCEL)
    return T
```

Al principio del algoritmo se calculan todas las distancias en el grafo $G = (V, A)$: SPL = { Matriz de distancias $d_{uv} \mid \forall\ u,v\ \epsilon\ V$, $d_{uv} = dist(u, v)$ }. Y a partir de estos cálculos, se buscará aquella arista que divida al grafo en dos conjuntos, $V1$ y $V2$, que tengan proximidad de vértices y caminos cortos en su totalidad. Un elemento utilizado en el algoritmo es el DG, el camino más largo entre un vértice u y un conjunto de vértices V, que tiene como código en SAGE:

```
def DG(u,V,SPL,DCEL):
    max = 0
    for i in V:
        if max < SPL[u][i]:
            max = SPL[u][i]
    return max
```

Aunque se divida el grafo en dos partes, $V1$ y $V2$, seguirán siendo grafos y no árboles generadores. Por ello, se deben crear dos árboles, cada uno dependiendo del conjunto dado, centrados cada uno en un extremo de la arista seleccionada anteriormente y formados por caminos mínimos. Existe un algoritmo que hace exactamente lo anterior: Dijkstra.

El algoritmo de Dijkstra [13] resuelve el problema de encontrar el árbol de mínimo camino. Consiste en encontrar un conjunto de aristas que conectan todos los vértices y minimizan las longitudes de los caminos desde el vértice raíz hasta todos los demás vértices del conjunto. El algoritmo es:

1. Se empieza con un vértice raíz, al que asignamos el camino de tamaño 0, en un grupo $S1$ y el resto de vértices en $S2$.

2. De los vértices en $S1$ se buscan sus vecinos en $S2$: $VecS1$.

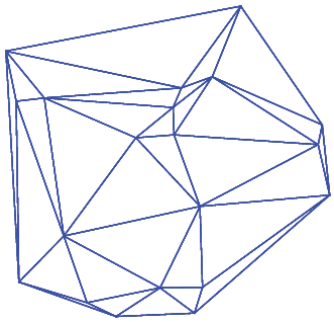
3. Calculando los caminos desde el vértice raíz a los elementos de $VecS1$, se guarda el nodo u que crea el camino más pequeño. Al árbol se añade la arista que conecta el nodo u con su respectivo vecino del grupo $S1$. Se borra u de $S2$ y se añade a $S1$.

4. Si en $S1$ estan todos los vértices, se para el algoritmo. En otro caso, volver al paso 2.

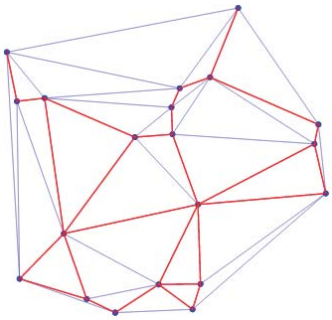
Para finalizar con el cálculo del MDST se unen los dos árboles con la arista que une x con y y se verifica que el árbol final T no repite aristas ni contiene ciclos.

Este algoritmo tiene complejidad computacional temporal $O(a * nlogn)$, siendo $a = |A|$ y $n = |V|$, porque: obtener todos los caminos del grafo es del orden $O(n * a + n^2 logn)$, ordenar los valores es $O(nlogn)$ y calcular los árboles de caminos cortos con el algoritmo de Dijkstra tiene una complejidad en el tiempo del orden $O(a + nlongn)$.

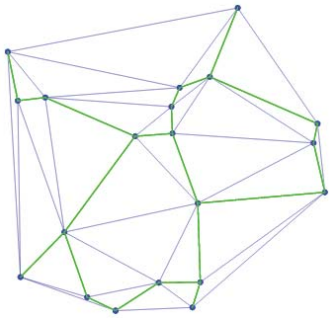
A continuación se mostrarán ejemplos del MDST con la triangulación de Delaunay, el grafo de Gabriel y el grafo de vecindad relativa:



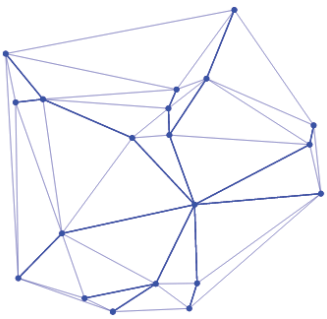
(a) Delaunay



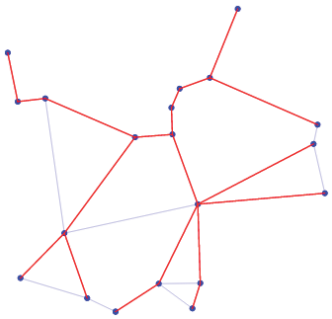
(b) Gabriel



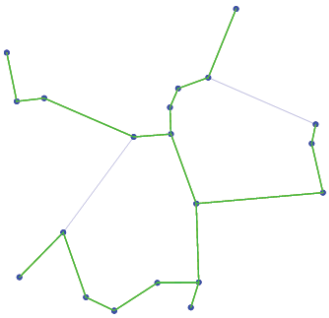
(c) RNG



(d) MDST deDelaunay



(e) MDST de Gabriel



(f) MDST del RNG

3.4. Conclusiones de los parámetros

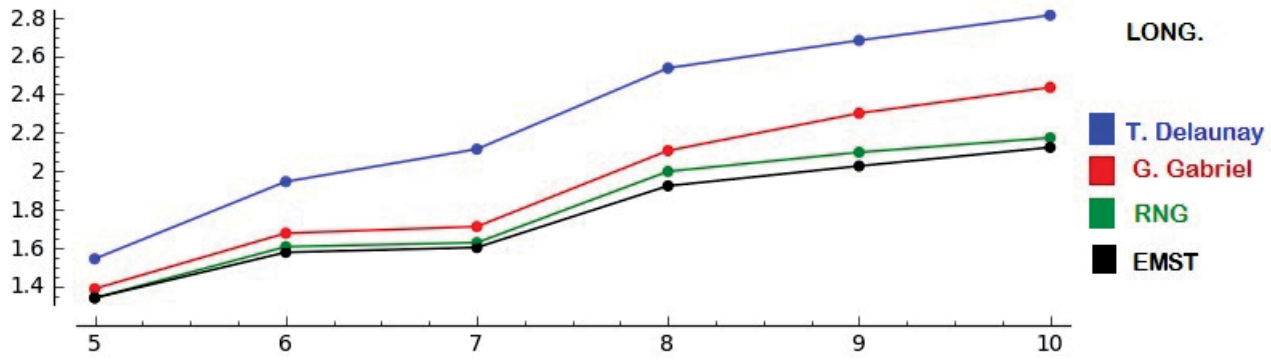
Utilizando la propiedad de comienzo de capítulo se sacan estas conclusiones:

- Según la cantidad de aristas el orden sería: $T.Delaunay > G.Gabriel > RNG > EMST$.

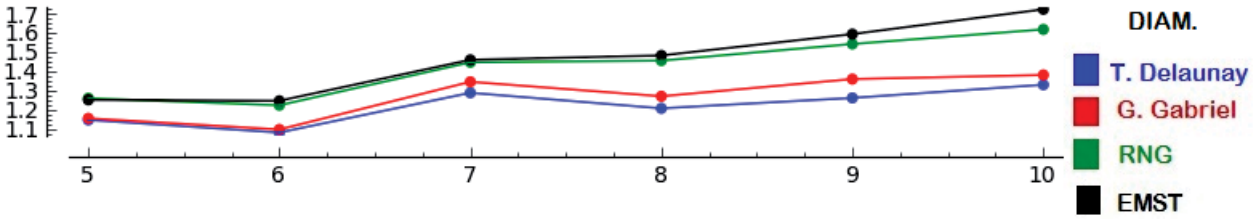
- En el *Capítulo 1* averiguamos que el diámetro es muy sensible a la cantidad de aristas del grafo. Por tanto, el valor del diámetro tendría el siguiente orden: $EMST > RNG > G.Gabriel > T.Delaunay$. Porque a más aristas más opciones de caminos y más árboles distintos donde elegir.
- El valor de la longitud debería ser: $T.Delaunay > G.Gabriel > RNG > EMST$.

¿Son todas estas observaciones ciertas? La primera de todas lo es por la definición de contenido de la propiedad anteriormente mencionada.

Para observar las afirmaciones se harán cálculos y gráficas de los cuatro árboles (el respectivo MDST de la t. de Delaunay, del g. de Gabriel y del RNG).



En la anterior gráfica está representada la longitud. Cada punto es la media de realizar 10 veces el cálculo. Se puede observar que los valores siguen la inecuación que se había razonado: $T.Delaunay > G.Gabriel > RNG > EMST$. Los valores del RNG y del EMST son muy cercanos entre ellos mientras que la triangulación de Delaunay tiene los resultados más grandes con un gran margen con en grafo de Gabriel.



Cada punto es la media de 30 valores del diámetro. En esta ocasión se ha subido el número de repeticiones para obtener más precisión ya que el rango de valores del diámetro es mucho más grande que el de la longitud. Por tanto, podría haber puntos en la gráfica que se salieran del esquema del resto de valores. Y, aún con esta precisión, hay valores muy cercanos de un grafo a otro; pero se podrían hacer dos separaciones: T. Delaunay con el grafo de Gabriel y RNG con el EMST. Estos dos

conjuntos tienen valores muy próximos entre ellos.
Los valores del diámetro siguen el esquema: $EMST > RNG > G.Gabriel > T.Delaunay$.

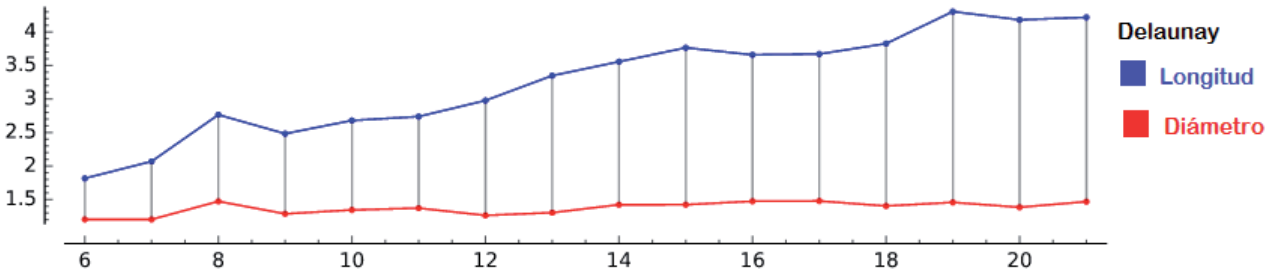
Capítulo 4

LONGITUD Y DIÁMETRO

En este capítulo se calculará, en primer lugar, la longitud y el diámetro en los grafos de los capítulos anteriores para estudiar ambos parámetros y saber que relación o comportamiento tienen en cada uno de los grafos geométricos con una misma nube de puntos aleatorios. A continuación, se hablará de las conclusiones de los resultados y por último unas herramientas extras para poder hacer cálculos sin los inconvenientes de trabajar en línea (Online).

4.1. Resultados

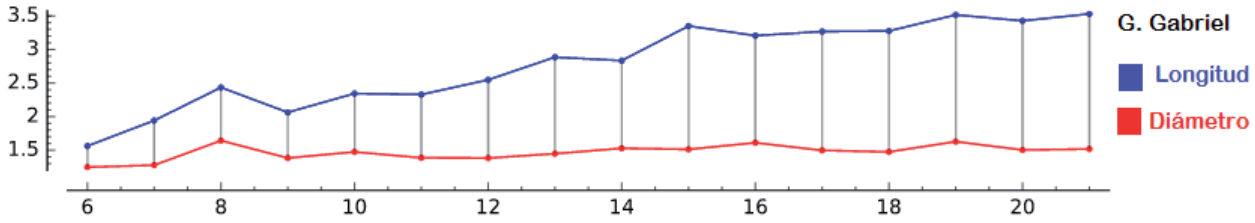
Triangulación de Delaunay. En este trabajo es el grafo con más aristas y, por tanto, el que lleva más tiempo computacionalmente de calcular su MDST. Tiene el diámetro más pequeño pero una medida de la longitud muy grande.



Los valores del diámetro crecen despacio aún añadiendo más puntos al plano; y la longitud alcanza medidas más y más grandes. La diferencia entre ambos parámetros aumenta bastante y se han necesitado muchas horas de computación para hacer estos cálculos.

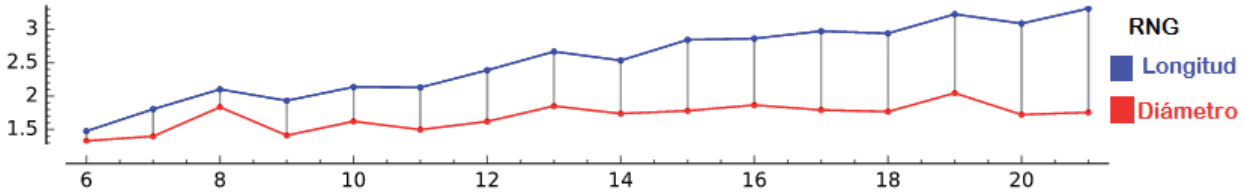
Grafo de Gabriel. Es un grafo de proximidad (como el RNG y el EMST), es decir, es un grafo geométrico definido por una propiedad P en el que cada par de vértices cumplen, uno de ellos o ambos vértices, la propiedad P. Está contenido en

la triangulación de Delaunay pero tiene peor diámetro, al haber menos aristas y, por tanto, menos atajos por el que puede haber caminos, y una longitud algo más pequeña.



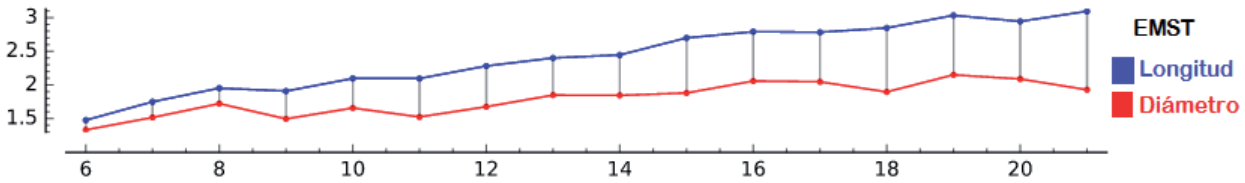
Comparándolo con la gráfica de la triangulación de Delaunay se observa un aumento en los valores del diámetro y una ligera disminución en la estimación de la longitud. Lo que lleva a una diferencia entre ambos parámetros más pequeña que la de Delaunay.

Grafo de Vecindad Relativa. Contenido en el grafo de Gabriel y creado a partir de la triangulación de Delaunay, es un grafo de proximidad con un diámetro mejor que el del EMST pero peor que el de Gabriel y una longitud próxima al del EMST y mejor que el del grafo de Gabriel.



La diferencia entre ambos parámetros se ha acortado pero a costa de empeorar la medida del diámetro.

Árbol generador de mínima longitud. Es el elemento más rápido de calcular computacionalmente y el único, en este trabajo, que usa una estructura distinta al DCEL. El EMST se construye a partir del algoritmo de Prim dando lugar a un árbol con los peores valores del diámetro, entre los otros grafos de proximidad y Delaunay, pero con la mejor estimación de la longitud.



Notablemente se observa un aumento en el diámetro pero una gran mejora en la longitud. Este árbol tiene la diferencia entre ambos parámetros más pequeña en el grupo de grafos obtenidos en este trabajo.

4.2. Conclusiones

El árbol generador euclídeo de longitud y diámetros cortos no tiene una solución única y óptima pues ambos parámetros son complementarios: mejorar (empeorar) la longitud empeorará (mejorará) el diámetro. Por lo tanto, dependiendo del objetivo de la construcción del árbol se seleccionará un árbol cercano a uno de los obtenidos anteriormente:

- **Mejor longitud.** Sin lugar a dudas habría que elegir el árbol generador de mínima longitud (EMST).
- **Mejor diámetro.** A pesar del tiempo de computación, el mejor árbol (o uno parecido a él) es el creado a partir de la triangulación de Delaunay pues al haber más aristas se pueden seleccionar aquellas que crean el camino más largo pero a la vez minimizándolo.
- **La diferencia entre ambos parámetros más corta posible.** En este caso se buscaría un árbol parecido al EMST o inclusive él mismo.
- **Un árbol que no empeore mucho los valores ni del diámetro ni de la longitud.** Sería un árbol parecido al MDST creado a partir del RNG.

Al completar el objetivo de estudiar la longitud y los diámetros en distintos árboles generadores geométricos se han encontrado formas de acercarse al árbol ideal de longitud y diámetros cortos con un desarrollo computacional simple y económico en tiempo y recursos.

4.3. Sage local y Python

SAGE es un entorno de cálculos matemáticos [14] basado en el lenguaje Python y de código abierto. Hay tres formas de acceder a SAGE:

- Servicio web SAGE disponible en Internet.
- Servidor web SAGE disponible en una red de área local.
- Servicio web SAGE disponible en la misma computadora en la que corre el navegador.

Este trabajo se construyó, en su gran medida, usando *SageMath Online* (*SAGE Cloud*). Pero, llegado al punto de realizar cálculos de mucho tiempo de ejecución, se tuvo que buscar nuevas formas o estructuras para trabajar porque *SageMath Online* tiene tiempo de refresco, lo cuál no permite la finalización de los cálculos.
SAGE tiene una estructura local, disponible en su página web, con la que poder trabajar. En un sistema operativo Windows es necesario instalar una máquina virtual y abrir el archivo descargable. Su inconveniente es que trabajar desde una máquina

virtual es más lento y pasar el código de un servidor a otro es complicado. Otra forma más sencilla es usar el sistema operativo de Linux: Ubuntu. Para instalar SAGE local se puede hacer de forma manual, yendo a la páquina web, o aprovechando las propiedades de Ubuntu para instalar archivos de una forma automática y sencilla escribiendo las siguientes líneas de código en la terminal:

```
sudo -E apt-add-repository -y ppa:aims/sagemath
sudo -E apt-get update
sudo -E apt-get install sagemath-upstream-binary
```

Con estos comandos se crea un directorio para guardar los archivos, se actualizan los elementos necesarios para SAGE y, por último, se instala. Escribiendo **./sage** en la terminal se creará un entorno de trabajo.

La última opción es trabajar en el entorno base de SAGE: Python. Los inconvenientes son los paquetes no instalados automáticamente, la complejidad de hacer gráficas y pequeños cambios en el lenguaje de programación (por ejemplo, en SAGE se usa ^ para las potencias y en Python **). Un error muy común es usar **x**(1/2)** para hacer raíces en Python, hay que usar **sqrt(x)**. Los paquetes necesarios para que funcione el código en Python son:

```
import scipy as sp
import numpy as np
from numpy.linalg import norm
from copy import deepcopy
from math import sqrt
import matplotlib.pyplot as plt
from random import random
```

Bibliografía

[1] William A. Stein (2005, Feb. 24). *Sage Documentation* (v7.1) [Online]. Available: http://doc.sagemath.org/html/en/index.html.

[2] J. García Miranda (2005). *Introducción a la teoría de grafos* [Online]. Available: http://www.ugr.es/~jesusgm/Curso%202005-2006/Matematica%20Discreta/Grafos.pdf.

[3] Fernando Berzal. *Recorrido sobre grafos*[Online]. Available: http://elvex.ugr.es/decsai/algorithms/slides/5%20Grafos.pdf.

[4] G. Hernández (2010, Dec. 2). *Teoría de Grafos* [Online]. Available: ftp://ftp.ing-mat.udec.cl/pub/ing-mat/asignaturas/525412/apuntes/Intro02.PDF.

[5] M. Abellanas, G. Hernández, J. L. Moreno, S. Ordóñez, V. Sacristán (2009, June 19). *Diagramas de Voronoi de alcance limitado* [Online]. Available: http://www-ma2.upc.edu/~geoc/DVALon/MemoriaDVALon.pdf.

[6] David Eppstein (1996, May 8). *Spanning Trees and Spanners*. Dept. Information and Computer Science. University of California, Irvine, CA 92717.

[7] M. Abellanas,U. Berzal,J. de Diego (2001, July 24). *Intercambio de aristas en grafos geométricos*[Online]. Available: http://www.dma.fi.upm.es/personal/mabellanas/tfcs/flips/Intercambios/html/teoria/teoria_del_esp.htm.

[8] L. M. Arteaga, G. Hernández (2002, July). *Triangulación de Delaunay*[Online]. Available: http://www.dma.fi.upm.es/recursos/aplicaciones/geometria_computacional_y_grafos/web/triangulaciones/delaunay.html.

[9] P. López, C. Yañez, L. Pró, J. Alcalde (2005). *Métodos computacionales orientados a la obtención del árbol EMST: Un aporte al problema de Steiner* [Online]. Available: http://sisbib.unmsm.edu.pe/bibvirtualdata/publicaciones/risi/n3_2005/a05.pdf.

[10] B. Y. Wu and K.-M. Chao (2004). *Spanning trees and optimization problems*, Discrete Mathematics and its Applications, Chapman and Hall/CRC, Boca Raton, FL. MR 2004i:90008Zbl 1072.90047.

[11] Svetlana Stolpner (2004). *Algorithms* [Online]. Available: `http://www.cim.mcgill.ca/~sveta/cg/algos.html`.

[12] Jerzy W. Jaromczyk, Godfried T. Toussaint. *Relative Neighborhood Graphs and Their Relatives* [Online]. Available: `http://www-ma2.upc.es/geoc/proximity-survey.pdf`.

[13] Paul Jensen (1999, Feb. 3). *Minimal Spanning Tree and Shortest PathTree Problems*[Online]. Available: `http://www.me.utexas.edu/~jensen/exercises/mst_spt/mst_spt.html`.

[14] Héctor Yanajara Parra. *Manual de SAGE para principiantes*[Online]. Available: `http://www.sagemath.org/es/Manual_SAGE_principiantes.pdf`.

Este documento esta firmado por

Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
Fecha/Hora	Fri Jun 10 20:10:16 CEST 2016
Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
Numero de Serie	630
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)